



PySpark Installation Guide

讲师：轩宇



PySpark

Case : WordCount

Anaconda Installing

Installing Python Linux

pyspark Command

spark-submit Application



Welcome to Python.org x

Python Software Foundation [US] | <https://www.python.org>

Python PSF Docs PyPI Jobs Community

python™

Search GO Socialize Sign In

About Downloads Documentation Community Success Stories News Events

```
# Python 3: Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!

# Input, assignment
>>> name = input('What is your name?\n')
>>> print('Hi, %s.' % name)
What is your name?
Python
Hi, Python.
```

Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. [Whet your appetite](#) with our Python 3 overview.

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. [>>> Learn More](#)

Get Started Download Docs Jobs

DOWNLOAD ANACONDA NOW

Download for



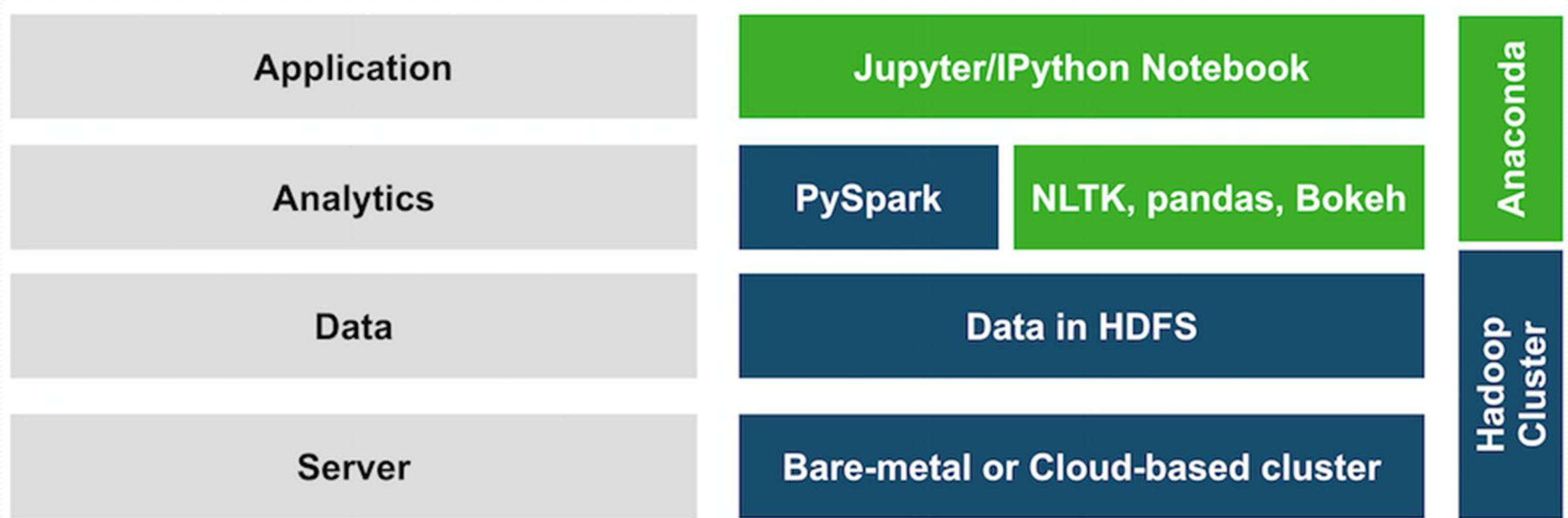
GET SUPERPOWERS WITH **ANACONDA**

Anaconda is the leading open data science platform powered by Python. The open source version of Anaconda is a high performance distribution of Python

Which version should I download and install?

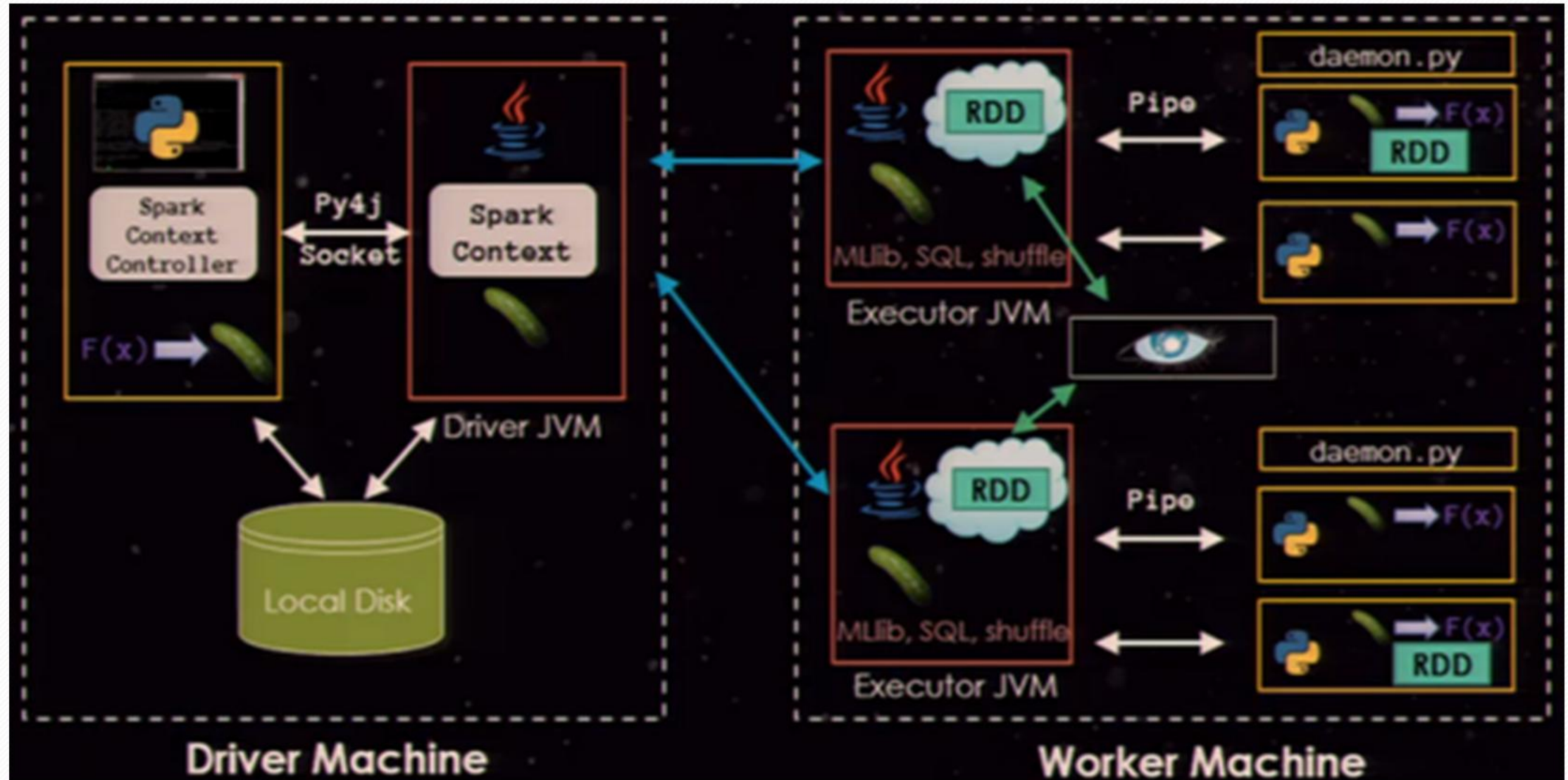
With Anaconda you can run multiple versions of Python in isolated environments, so choose the download with the Python version that you use more often, as that will be your default Python

USING ANACONDA WITH PYSPARK



<https://www.continuum.io/blog/developer-blog/using-anaconda-pyspark-distributed-language-processing-hadoop-cluster>

PYSPARK ARCHITECTURE



PYSPARK BASICS

Python For Data Science Cheat Sheet

PySpark Basics

Learn Python for data science interactively at www.datacamp.com



Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python



Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = "local[2]")
```

Inspect SparkContext

```
>>> sc.version
>>> sc.pythonVer
>>> sc.master
>>> str(sc.sparkHome)
>>> str(sc.sparkUser())
```

Retrieve SparkContext version
Retrieve Python version
Master URL to connect to
Path where Spark is installed on worker nodes
Retrieve name of the Spark User running

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()
>>> rdd.count()
>>> rdd.countByKey()
>>> rdd.countByValue()
>>> rdd.collectAsMap()
>>> rdd.sum()
>>> sc.parallelize([ ]).isEmpty()
```

List the number of partitions
Count RDD instances
Count RDD instances by key
Count RDD instances by value
Collect RDD as a map
Sum of RDD elements
Check whether RDD is empty

Summary

```
>>> rdd.max()
>>> rdd.min()
>>> rdd.mean()
>>> rdd.stdev()
>>> rdd.variance()
>>> rdd.histogram()
```

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins

Reshaping Data

Reducing

```
>>> rdd.reduce(lambda x, y: x+y)
>>> rdd.reduce(lambda a, b: a + b)
```

Merge the rdd values for each key
Merge the rdd values

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)
>>> rdd3.groupByKey()
>>> rdd3.mapValues(list)
>>> rdd3.mapValues(lambda x: x[0])
```

Return RDD of grouped values
Group rdd by key

Aggregating

```
>>> seqOp = (lambda x, y: x[0]+y, x[1]+1)
>>> combOp = (lambda x, y: x[0]+y[0], x[1]+y[1])
>>> rdd3.aggregate((0,0), seqOp, combOp)
>>> rdd3.aggregateByKey((0,0), seqOp, combOp)
>>> rdd3.fold(0, add)
>>> rdd3.foldByKey(0, add)
>>> rdd3.mapValues(lambda x: x*x)
```

Aggregate RDD elements of each partition and then the results
Aggregate values of each RDD key
Aggregate the elements of each partition, and then the results
Merge the values for each key
Create tuple of RDD elements by

Check Out PySpark Cheat Sheet

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize(['a', 'b', 'c', 'd'])
>>> rdd2 = sc.parallelize(['a', 'b', 'c', 'd'], 4)
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize(['a', 'b', 'c', 'd'], 4)
```

External Data

Read either one text file from HDFS, a local file system or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("my/directory/")
```

Selecting Data

Getting

```
>>> rdd.collect()
>>> rdd.take(2)
>>> rdd.first()
>>> rdd.top(2)
>>> rdd.first()
```

Return a list with all RDD elements
Take first 2 RDD elements
Take first RDD element
Take top 2 RDD elements
Take first RDD element

Sampling

```
>>> rdd.sample(False, 0.15, 81).collect()
```

Return sampled subset of rdd

Filtering

```
>>> rdd.filter(lambda x: "a" in x)
>>> rdd.distinct().collect()
>>> rdd.keys().collect()
```

Filter the RDD
Return distinct RDD values
Return (key,value) RDD's keys

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)
>>> rdd.foreachPartition(g)
```

Apply a function to all RDD elements

Sort

```
>>> rdd.sortBy(lambda x: x[1])
>>> rdd.sortByKey()
>>> rdd.sortBy(lambda x: x[1])
```

Sort RDD by given function
Sort (key, value) RDD by key

Repartitioning

```
>>> rdd.repartition(4)
>>> rdd.coalesce(1)
```

New RDD with 4 partitions
Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost:port/child",
>>>                        "org.apache.hadoop.mapred.TextInputFormat")
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

